Введение

Дисциплина «Математическая логика и теория алгоритмов» закладывает прочный фундамент для изучения практически всех специализированных курсов, преподаваемых в технических вузах и университетах для направлений подготовки, связанных с изучением компьютеров, информационных технологий и коммуникационных сетей.

Непосредственная цель дисциплины «Математическая логика и теория алгоритмов» — дать основы теории математического обеспечения для современных компьютерных и информационных технологий.

Практически половину курса «Математическая логика и теория алгоритмов» занимает теория алгоритмов: подходы к понятию алгоритма, свойства, типы алгоритмических моделей, вопросы оптимизации программ, математические основы программирования.

В настоящее время многие вопросы теории алгоритмов, особенно их эффективность и сложность, рассматриваются и являются важными при решении многих практических задач. В связи с развитием сферы применимости теории алгоритмов возрастает значение и актуальность этого раздела в изучаемой дисциплине. Теория алгоритмов является мощным инструментом для исследований во множестве новых областей. Основные положения теории алгоритмов приведены в части 2 данного учебно-методического пособия.

В рамках данного раздела дисциплины предполагается изучить элементы теории алгоритмов и их сложности.

1. Определение сложности алгоритмов

Понятие «алгоритм» давно является привычным не только для математиков. Оно является концептуальной основой разнообразных процессов обработки информации. Возможность автоматизации таких процессов обеспечивается наличием соответствующих алгоритмов. С алгоритмами первое знакомство происходит в начальной школе при изучении арифметических действий с натуральными числами. В упрощенном понимании «алгоритм» — это то, что можно запрограммировать на компьютере.

Слово «алгоритм» содержит в своем составе преобразованное географическое название Хорезм. Термин «алгоритм» обязан своим происхождением великому ученому средневекового Востока — Муххамад ибн Муса ал-Хорезми (Магомет, сын Моисея, из Хорезма). Он жил приблизительно с 783 по 850 гг., и в 1983 г. отмечалось 1200-летие со дня его рождения в городе Ургенче — областном центре современной Хорезмской области Узбекистана. В латинских переводах с арабского арифметического трактата ал-Хорезми его имя транскрибировалось как cugorismi. Откуда и пошло слово «алгоритм» — сначала для обозначения алгоритмов цифровых вычислений десятичной позиционной арифметики, а затем для обозначения произвольных процессов, в которых искомые величины решаемых задач находятся последовательно из исходных данных по определенным правилам и инструкциям.

Зародившись еще в средневековье, интуитивное понятие алгоритма с середины XX в. получило широкое распространение. Однако формирование строгого научного понятия алгоритма не закончено и в настоящее время. Прежде, чем затронуть эту сложную и важную проблему, перейдем к примерам.

Рассмотрим алгоритм Евклида, который допускает «арифметическую» интерпретацию — нахождение наибольшего общего делителя (НОД) двух натуральных чисел. Если дана пара натуральных чисел $a > b$, то результат деления a на b с остатком r эквивалентен равенству $a = q \cdot b + r$. В этом случае НОД(a,b) = НОД(b,r), (1) например, $21 = 2 \cdot 9 + 3$ и НОД(21,9) = = НОД(9,3) = 3. Пусть a_0 и a_1 — натуральные числа, $a_0 > a_1$.

Поиск НОД(a_0, a_1) по алгоритму Евклида требует выполнения серии однотипных шагов, каждый из которых — деление с остатком:

$$\begin{aligned} a_0 &= q_1 a_1 + a_2, \\ &\downarrow \quad \downarrow \\ a_1 &= q_2 a_2 + a_3, \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n-2} &= q_{n-1} a_{n-1} + a_n, \\ &\downarrow \quad \downarrow \\ a_{n-1} &= q_n a_n. \end{aligned} \qquad (1)$$

В этом случае $a_n = НОД(a_0, a_1)$, так как

$$НОД(a_0, a_1) = НОД(a_1, a_2) = \ldots = НОД(a_{n-1}, a_n) = a_n .$$

Формула (1) позволяет оформить алгоритм Евклида в виде рекурсивной процедуры, то есть процедуры, вызывающей самое себя (пример простейшей рекурсии — факториал: $n! = (n-1)!\,n$). Запишем процедуру *NOD* на псевдокоде, близком языку Паскаль (заменив для краткости блочные скобки begin, end на фигурные: {, }):

```
NOD(a,b)
{ if b=0 then NOD:= a
else { r:= a mod b; NOD:= NOD(b,r) }}
```

Здесь использована операция *mod* вычисления остатка от деления *a* на *b* (например, 30 mod 21 = 9). При вычислении NOD(30,21) = NOD(21,9) = NOD(9,3) = 3 процедура вызывает себя трижды.

В численном анализе близкими к рекурсиям являются рекуррентные соотношения, оформленные в виде итерационных схем:

$$\begin{cases} x_{n+1} = \Phi(x_n), \\ x_0 - \text{задано}, \end{cases}$$

решающих уравнения типа $x = \Phi(x)$, где x — вещественное число (x может быть и вектором с вещественными координатами). x_n — последовательность рациональных приближений, при определенных условиях сходящаяся к точному решению.

Итерации проводятся до тех пор, пока расстояние между соседними приближениями не станет меньше заданной погрешности: $|x_n - \Phi(x_n)| < \varepsilon$. Здесь возникает проблема зацикливания алгоритма, если последовательность x_n расходится. Рассмотрим алгоритм $A(x_0)$, решающий уравнение $x = x_2$ методом итераций:

```
A(x0)
{ x:= x0;
while abs(x–x*x) > 1e–6 do x:= x*x;
A:= x }
```

Вызвав процедуру $A(0{,}99)$, получим результат, приближенно равный нулю — меньшему корню уравнения, так как $x_n = 0{,}99^m$ стремится к нулю с ростом степени $m = 2^n$. Но незначительное увеличение исходного данного x_0 нашего алгоритма всего лишь на 0,02 приведет к расходящемуся процессу, так как $1{,}01^m \to \infty$ при $m = 2^n \to \infty$.

Численные методы также можно «арифметизировать» — свести к алгоритмам, работающим с натуральными числами: из-за ограниченности разрядной сетки компьютера вещественные числа — бесконечные десятичные дроби — заменяются рациональными приближениями с конечной мантиссой, а любое рациональное число можно задать парой натуральных чисел, например:

$$1{,}5 = 3 : 2 \sim (3, 2); \; -2 = 3 - 5 \sim (3, 5).$$

С кибернетической точки зрения алгоритм можно рассматривать, как «техническое» устройство, преобразующее набор входных данных $x_1, x_2, \ldots x_m$ в выходные параметры $y_1, y_2, \ldots y_n$, например, как процедуру умножения матрицы на вектор (*рис. 1*):

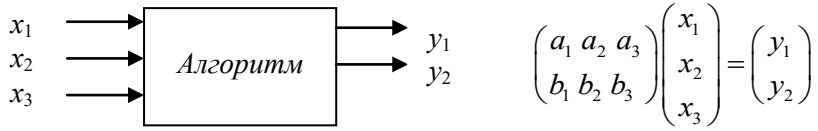

Рис. 1. Алгоритм как «техническое» устройство

Чтобы ограничиться функциями, принимающими только числовые, а не векторные значения, мы можем данный пример представить в виде двух алгоритмов:

$$A(x) = a_1x_1 + a_2x_2 + a_3x_3;$$
$$B(x) = b_1x_1 + b_2x_2 + b_3x_3.$$

Приведенные соображения позволяют свести теоретические проблемы существования и сходимости алгоритмов к изучению натуральных функций натуральных аргументов: A: $N \times N \times \ldots \times N \to N$, где N — множество натуральных чисел.

Вплоть до 30-х гг. нашего столетия понятие алгоритма оставалось интуитивно понятным, имевшим скорее методологическое, а не математическое значение. Так, к началу XX в. много ярких примеров дала алгебра и теория чисел. Среди них упомянем алгоритм Евклида нахождения наибольшего общего делителя двух натуральных чисел или двух целочисленных многочленов, алгоритм Гаусса решения системы линейных уравнений над полем, алгоритм нахождения рациональных корней многочленов одного переменного с рациональными коэффициентами, алгоритм Штурма определения числа действительных корней многочлена с действительными коэффициентами на некотором отрезке действительных чисел, алгоритм разложения многочлена одного переменного над конечным полем на неприводимые множители.

Указанные алгоритмические проблемы решены путем указания конкретных разрешающих процедур. Для получения результатов такого типа достаточно интуитивного понятия алгоритма. Однако в начале XX в. были сформулированы алгоритмические проблемы, положительное решение которых представлялось маловероятным.

Решение таких проблем потребовало привлечения новых логических средств. Ведь одно дело доказать существование разрешающего алгоритма — это можно сделать, используя интуитивное понятие алгоритма. Другое дело — доказать несуществование алгоритма — для этого нужно знать точно — что такое алгоритм.

Задача точного определения понятия алгоритма была решена в 30-х гг. в работах Гильберта, Черча, Клини, Поста, Тьюринга в двух формах: на основе понятия рекурсивной функции и на основе описания алгоритмического процесса. Рекурсивная функция это функция, для которой существует алгоритм вычисления ее значений по произвольному значению аргумента. Класс рекурсивных функций был определен строго как конкретный класс функций в некоторой формальной системе. Был сформулирован тезис (называемый «тезис Черча»), утверждающий, что данный класс функций совпадает с множеством функций, для которых имеется алгоритм вычисления значений по значению аргументов.

Другой подход заключался в том, что алгоритмический процесс определяется как процесс, осуществимый на конкретно устроенной машине (называемой «машиной Тьюринга»). Был сформулирован тезис (называемый «тезис Тьюринга»), утверждающий, что любой алгоритм может быть реализован на подходящей машине Тьюринга (МТ).

Оба данных подхода, а также другие подходы (Марков, Пост) привели к одному и тому же классу алгоритмически вычислимых функций и подтвердили целесообразность использования тезиса Черча или тезиса Тьюринга для решения алгоритмических проблем. Поскольку понятие рекурсивной функции строгое, то с помощью обычной математической техники можно доказать, что решающая некоторую задачу функция не является рекурсивной, что эквивалентно отсутствию для данной задачи разрешающего алгоритма.

Аналогично, несуществование разрешающей МТ для некоторой задачи равносильно отсутствию для нее разрешающего алгоритма. Указанные результаты составляют основу так называемой дескриптивной теории алгоритмов, основным содержанием которой является классификация задач по признаку алгоритмической разрешимости, т. е. получение высказываний типа «Задача Р алгоритмически разрешима» или «Задача Р алгоритмически неразрешима». В данном направлении получен ряд фундаментальных результатов. Среди них отрицательное решение Новиковым П. С. в 1952 г. классической проблемы тождества для конечно определенных групп, сформулированной Деном в 1912 г. Знаменитая десятая проблема Гильберта, сформулированная им в 1900 г. (среди других 23 проблем) формулируется так: «10. Задача о разрешимости диофантова уравнения. Пусть задано диофантово уравнение с произвольными неизвестными и целыми рациональными числовыми коэффициентами. Указать способ, при помощи которого возможно после конечного числа операций установить, разрешимо ли это уравнение в целых рациональных числах». Алгоритмическая неразрешимость 10-й проблемы Гильберта была доказана в 1970 г. Мятиясевичем Ю. В.

В настоящее время теория алгоритмов образует теоретический фундамент вычислительных наук. Применение теории алгоритмов осуществляется как в использовании самих результатов (особенно это касается использования разработанных

алгоритмов), так и в обнаружении новых понятий и уточнении старых. С ее помощью проясняются такие понятия как доказуемость, эффективность, разрешимость, перечислимость и другие.

В технику термин «алгоритм» пришел вместе с кибернетикой. Понятие алгоритма помогло, например, точно определить, что значит эффективно задать последовательность управляющих сигналов. Применение компьютеров послужило стимулом развитию теории алгоритмов и изучению алгоритмических моделей, к самостоятельному изучению алгоритмов с целью их сравнения по рабочим характеристикам (числу действий, расходу памяти), а также их оптимизации.

Возникло важное направление в теории алгоритмов — сложность алгоритмов и вычислений. Начала складываться так называемая метрическая теория алгоритмов, основным содержанием которой является классификация задач по классам сложности. Сами алгоритмы стали объектом точного исследования, как и те объекты, для работы с которыми они предназначены. В этой области естественно выделяются задачи получения верхних и задачи получения нижних оценок сложности алгоритмов, и методы решения этих задач совершенно различны.

Для получения верхних оценок достаточно интуитивного понятия алгоритма. Для этого строится неформальный алгоритм решения конкретной задачи, и затем он формализуется для реализации на подходящей алгоритмической модели. Если показывается, что сложность (время или память) вычисления для этого алгоритма не превосходит значения подходящей функции при всех значениях аргумента, то эта функция объявляется верхней оценкой сложности решения рассматриваемой задачи. В области получения верхних оценок получено много ярких результатов для конкретных задач. Среди них разработаны быстрые алгоритмы умножения целых чисел, многочленов, матриц, решения линейных систем уравнений, которые требуют значительно меньше ресурсов, чем традиционные алгоритмы.

Установить нижнюю оценку — значит доказать, что никакой алгоритм вычисления не имеет сложности меньшей, чем заданная граница. Для получения результатов такого типа необходима точная фиксация рассматриваемой алгоритмической модели, и такие результаты получены только в очень жестких вычислительных моделях. В связи с этим получила развитие

проблематика получения «относительных» нижних оценок, так называемая теория *NP*-полноты, связанная с труднорешаемостью переборных задач.

Основные требования к алгоритмам

1. Каждый алгоритм имеет дело с данными — входными, промежуточными, выходными. Для того чтобы уточнить понятие данных, фиксируется конечный алфавит исходных символов (цифры, буквы и т. п.) и указываются правила построения алгоритмических объектов. Типичным используемым средством является индуктивное построение. Например, определение идентификатора в Object Pascal: идентификатор — это либо буква, либо идентификатор, к которому приписана справа либо буква, либо цифра. Слова конечной длины в конечных алфавитах — наиболее обычный тип алгоритмических данных, а число символов в слове — естественная мера объема данных. Другой случай алгоритмических объектов — формулы. Примером могут служить формулы алгебры предикатов и алгебры высказываний. В этом случае не каждое слово в алфавите будет формулой.

2. Алгоритм для размещения данных требует памяти. Память обычно считается однородной и дискретной, т. е. она состоит из одинаковых ячеек, причем каждая ячейка может содержать один символ данных, что позволяет согласовать единицы измерения объема данных и памяти.

3. Алгоритм состоит из отдельных элементарных шагов, причем множество различных шагов, из которых составлен алгоритм, конечно. Типичный пример множества элементарных шагов — система команд компьютера.

4. Последовательность шагов алгоритма детерминированная, т. е. после каждого шага указывается, какой шаг следует выполнять дальше, либо указывается, когда следует работу алгоритма считать законченной.

5. Алгоритм должен обладать результативностью, т. е. останавливаться после конечного числа шагов (зависящего от исходных данных) с выдачей результата. Данное свойство иногда называют сходимостью алгоритма.

6. Алгоритм предполагает наличие механизма реализации, который по описанию алгоритма порождает процесс вычисления на основе исходных данных. Предполагается, что описание алгоритма и механизм его реализации конечны.

Можно заметить аналогию с вычислительными машинами. Требование 1 соответствует цифровой природе компьютера, требование 2 — памяти компьютера, требование 3 — программе машины, требование 4 — ее логической природе, требования 5, 6 — вычислительному устройству и его возможностям.

Имеются также некоторые черты неформального понятия алгоритма, относительно которых не достигнуто окончательного соглашения. Эти черты сформулируем в виде вопросов и ответов.

7. Следует ли фиксировать конечную границу для размера входных данных?

8. Следует ли фиксировать конечную границу для числа элементарных шагов?

9. Следует ли фиксировать конечную границу для размера памяти? Следует ли ограничить число шагов вычисления?

На все эти вопросы далее принимается ответ «НЕТ», хотя возможны и другие варианты ответов, поскольку у физически существующих компьютеров соответствующие размеры ограничены. Однако теория, изучающая алгоритмические вычисления, осуществимые в принципе, не должна считаться с такого рода ограничениями, поскольку они преодолимы, по крайней мере, в принципе (например, вообще говоря, любой фиксированный размер памяти всегда можно увеличить на одну ячейку).

Таким образом, уточнение понятия алгоритма связано с уточнением алфавита данных и формы их представления, памяти и размещения в ней данных, элементарных шагов алгоритма и механизма реализации алгоритма. Однако эти понятия сами нуждаются в уточнении. Ясно, что их словесные определения потребуют введения новых понятий, для которых в свою очередь снова потребуются уточнения и т. д. Поэтому в теории алгоритмов принят другой подход, основанный на конкретной алгоритмической модели, в которой все сформулированные требования выполняются очевидным образом.

При этом используемые алгоритмические модели универсальны, т. е. моделируют любые другие разумные алгоритмические модели, что позволяет снять возможное возражение против такого подхода: не приводит ли жесткая фиксация алгоритмической модели к потере общности формализации алгоритма? Поэтому данные алгоритмические модели отождествляются

с формальным понятием алгоритма. В дальнейшем будут рассмотрены основные типы алгоритмических моделей, различающиеся исходными трактовками, что такое алгоритм.

Первый тип трактует алгоритм как некоторое детерминированное устройство, способное выполнять в каждый момент лишь строго фиксированное множество операций. Основной теоретической моделью такого типа является МТ, предложенная А. Тьюрингом в 30-х гг. и оказавшая существенное влияние на понимание логической природы разрабатываемых компьютеров. Другой теоретической моделью данного типа является машина произвольного доступа (МПД) — введенная достаточно недавно (в 70-х гг.) с целью моделирования реальных вычислительных машин и получения оценок сложности вычислений.

Второй тип связывает понятие алгоритма с традиционным представлением — процедурами вычисления значений числовых функций. Основной теоретической моделью этого типа являются рекурсивные функции — исторически первая формализация понятия алгоритма.

Третий тип алгоритмических моделей — это преобразования слов в произвольных алфавитах, в которых операциями являются замены кусков слов другим словом. Основной теоретической моделью этого типа являются нормальные алгоритмы Маркова.

Теория алгоритмов оказала существенное влияние на развитие компьютеров и практику программирования. В теории алгоритмов были предугаданы основные концепции, заложенные в аппаратуру и языки программирования. Упоминаемые выше главные алгоритмические модели математически эквивалентны; но на практике они существенно различаются сложностными эффектами, возникающими при реализации алгоритмов, и породили разные направления в программировании. Так, микропрограммирование строится на идеях машин Тьюринга, структурное программирование заимствовало свои конструкции из теории рекурсивных функций, языки символьной обработки информации (ПРОЛОГ, РЕФАЛ) берут начало от нормальных алгоритмов Маркова и систем Поста.

Существует точка зрения, что «алгоритмические концепции играют в процессе обучения и воспитания современного человека фундаментальную роль, сравнимую лишь с ролью письменности».

Конец ознакомительного фрагмента.
Приобрести книгу можно
в интернет-магазине
«Электронный универс»
e-Univers.ru